(A-69999/RMA)

SYSTEM AND METHOD FOR THIN PROCEDURAL MULTI-MEDIA PLAYER RUN-TIME ENGINE HAVING APPLICATION PROGRAM LEVEL COOPERATIVE MULTI-THREADING AND CONSTRAINED RESOURCE RETRY WITH ANTI-STALL FEATURES

I Claim:

1. A small low-overhead content playback engine comprising:

a main procedure implemented in portable code, native processor code or hardware blocks that executes cooperative player engine threads in turn;.

a boot-up sequence to assign an instruction input buffer to a startup thread, loads the first procedural multi-media player instructions, and starts the startup thread in a running state;

an instruction dispatcher that fetches each instruction word of a thread in sequence or as directed by branching instructions, and calls a native code function or hardware block to execute each instruction word and the parameters that follow it in turn;

a set of native code functions or hardware blocks which together carry out the functions of the multi-media player instruction words and parameters; and

a hardware extraction layer implemented in native code functions or hardware blocks that marry the portable portions of said player engine to the parts that are specific to the application or device that makes use of the player.

2. A method for a thin low-overhead multi-media procedural content player engine, said method comprising steps of:

receiving a file for playback comprising at least one sequence of fixed length words organized by having a plurality of instructions arranged as a linear sequence where parameters associated with a particular instruction immediately follow the particular instruction and wherein subsequent instructions follow the parameters associated with a previous instruction;

operating, by said playback engine, on the sequence of instructions and parameters, said operating including:

fetching the next word in the sequence, said word including an indicia of the function to be performed;

executing said identified function; and

when said identified function utilizes parameters, said function then: (i) fetching the parameters that follow the instruction; (ii) performing the instruction using the function and parameters; (iii) advancing a program counter past the parameters to the next instruction in the sequence; and, (iv) returning a status code for the instruction.

3. The method of claim 2, wherein said status code being selected from the set of status codes consisting of a success status code, an error status code, a yield status code, a informative status code, and a retry instruction status code.

4. The method of claim 2, wherein said instruction and parameters are arranged according to the scheme Instruction1, param1a, param1b, ..., Instruction2, param2a, param2b, param2c, ..., InstrutionN, paramNa, ..., paramNm.

5. The method in claim 2, wherein said content player comprises a StoryMail story player.

6. The method of claim 2, wherein said status code being selected from the set of status codes consisting of a success status code, an error status code, a yield status code, a informative status code, and a retry instruction status; and

said instruction and parameters are arranged according to the scheme Instruction1, param1a, param1b, ..., Instruction2, param2a, param2b, param2c, ..., InstrutionN, paramNa, ..., paramNm.; and

said content player comprises a StoryMail story player.

7. The method in claim 2, wherein said fixed length words being 32-bit words.

8. The method in claim 2, wherein said fixed length words being selected from the set of fixed length word sizes consisting of 8-bit words, 16-bit words, 32-bit words, 40-bit words, 64-bit words, 96-bit words, 128-bit words, 256-bit words, 512-bit words, and any other fixed length word or byte size.

9. The method in claim 2, wherein receiving a file for playback comprising at least one sequence of said fixed length words.

10. The method in claim 2, wherein said fixed length words and parameters are comprised of numeric and/or symbolic values in any combination.

11. The method in claim 2, wherein said instruction values identify individual functions within a library of functions.

12. The method in claim 11, wherein said instruction values identifies one or more branch instructions.

13. The method in claim 2, wherein said run-time module program(s) is thin.

14. The method in claim 2, wherein said run-time module program(s) is thin and implemented with fewer than about 200 lines of program code.

15. The method in claim 2, wherein said content comprises a StoryMail story.

16. The method in claim 2, wherein said run-time module program(s) is thin and implemented with fewer than about 100 lines of program code.

17. The method in claim 2, wherein said run-time module program(s) is thin and implemented with fewer than about 50 lines of program code.

18. The method in claim 2, wherein said run-time module program(s) is thin and implemented with fewer than about 50 lines of C language program code.

19. The method in claim 2, wherein said run-time module has a low-overhead relative to conventional run-time systems because no sophisticated parsing, threading, synchronization, memory allocation or garbage collection mechanisms are needed.

20. The method in claim 2, wherein execution speed is increased relative to conventional methods because processor intensive functions are performed with native processor code as part of an op-code's implementation, and all the control and navigation are performed in the very compact and very compressible story language instructions.

21. The method in claim 2, wherein said method is electrical power conservative because processor intensive functions are performed with optimized native processor code as part of an op-code's implementation, and all the control and navigation are performed in the very compact and very compressible story language instructions.

22. The method in claim 21 wherein said processor intensive functions include inverse discrete cosine transforms (IDCTs).

23. The method in claim 21, wherein said story language code is small.

24. The method in claim 2, wherein said run-time module program mechanism uses a common set of small functions over and over again to provide the functional capabilities of larger conventional programs so that tasks can be run within the data and code caches of at least some processors of conventional computers and information appliances.

25. The method in claim 21, wherein said method is performed with fewer layers of abstraction functional modules and less complex algorithms.

26. The method in claim 2, wherein said method provides a run-time system that eliminates the need to implement any of the following complex algorithm types: (i) thread creation and round robin thread scheduling with thread priority systems, (ii) native operating system or C library memory allocation functions, (iii) memory garbage collection functions, (iv) interrupt system functions, (v) picture decompression algorithms, (vi) multimedia playback system, (vii) user controls, and (viii) video and/or audio synchronization algorithms.

27. The method in claim 2, wherein the size of the native code to perform playback of multimedia application or messages in story format is no more than from about 30 kilobytes to about 300 kilobytes.

28. The method in claim 2, wherein the size of the native code to perform playback of multimedia application or messages in story format is no more than about 50 kilobytes.

29. The method in claim 2, wherein the size of the native code to perform playback of multimedia application or messages in story format is no more than about 100 kilobytes.

30. The method in claim 2, wherein the size of native code is reduced by a factor of about 100 as compared to conventional implementations.

31. The method in claim 2, wherein the size of native code is reduced by from by a factor of about 5 times to a factor of about 1000 times as compared to conventional implementations.

32. The method in claim 2, wherein the size of the native code to perform playback of multimedia application or messages in story format is less than 500 kilobytes.

33. The method in claim 2, wherein said run-time module provides cooperative multi-threading of various visual or audio special effects.

34. The method in claim 2, wherein said cooperative multi-threading occurs at the level of the application program.

35. The method in claim 2, wherein said cooperative multi-threading procedure further includes a constrained resource retry procedure.

36. The method in claim 2, wherein said cooperative multi-threading with constrained resource retry occurs at the level of the application program.

37. The method in claim 36, wherein said multi-threaded with constrained resource retry procedure includes steps of: running sequences of instructions for a thread as long as the instruction functions return as status code of success, and then executing the sequences of instructions for the next thread for as long as the instruction functions return a status code of success; a yield status code being returned for any instruction or sequence of instructions that takes more than a predetermined time to complete so that other threads and their instructions will have an opportunity to run.

38. The method in claim 37, wherein said status code is set to retry when a constrained resource blocks the execution of the instruction, thereby allowing other threads to run before the instruction is retried.

39. The method of claim 36, wherein said resource constraint is selected from the set of constrains consisting of: time being greater than some predetermined value, time being less than some predetermined value, time being equal to some predetermined value, a buffer being available, a buffer not being available, a variable being less than a predetermined value, a variable being greater than a predetermined value, a variable being equal to a predetermined value, a variable having any predetermined logical or arithmetic relation to a reference value, a hardware device being ready, a hardware device not being ready, an electronic communication or protocol having been completed, an electronic communication or protocol not having been completed, and combinations thereof.

40. The method in claim 39, wherein said method further provides thread or media playback synchronization.

41. The method in claim 40, wherein said thread synchronization including input, video playback, audio playback, special effects of video, special effects of audio, or combinations thereof.

42. The method in claim 39, wherein executing a "wait until time" type instruction that will start execution and/or not complete execution until a predetermined set time or set times.

43. The method in claim 42, wherein said wait until time instruction comprising a TIME_OP instruction.

44. The method in claim 43, wherein said set time being defined by a reference to a relative time, whether or not using indirection plus post operations, to an elapsed time difference, to an absolute time reference.

45. The method in claim 42, wherein said wait until time type instruction returning a retry instruction status if it is not time for the instruction to be executed and/or to complete execution, said return of said retry instruction status code causing execution of the next thread to execute.

46. The method in claim 45, wherein each time the "wait until time" instruction containing thread starts again it will retry the same instruction until the set time.

47. The method in claim 46, wherein said set time is a constrained resource.

48. The method in claim 47, wherein said constrained resource is time and the instruction constrained by time is retried if the time is not the set time or within some predetermined difference from the set time.

49. The method in claim 39, wherein a memory buffer is a constrained resource and an instruction that needs a memory buffer will return a retry instruction status code if the needed memory buffer is not available.

50. The method in claim 39, wherein use of said retry instruction status reducing the likelihood of stalling the processor as a result of a resource not being available when needed.

51. The method in claim 39, wherein synchronization of threads is achieved using a wait for flag in a wait until time instruction, said wait for flag comprising a variable which is itself an element of a memory buffer.